Coloring-based Coalescing for Graph Coloring Register Allocation

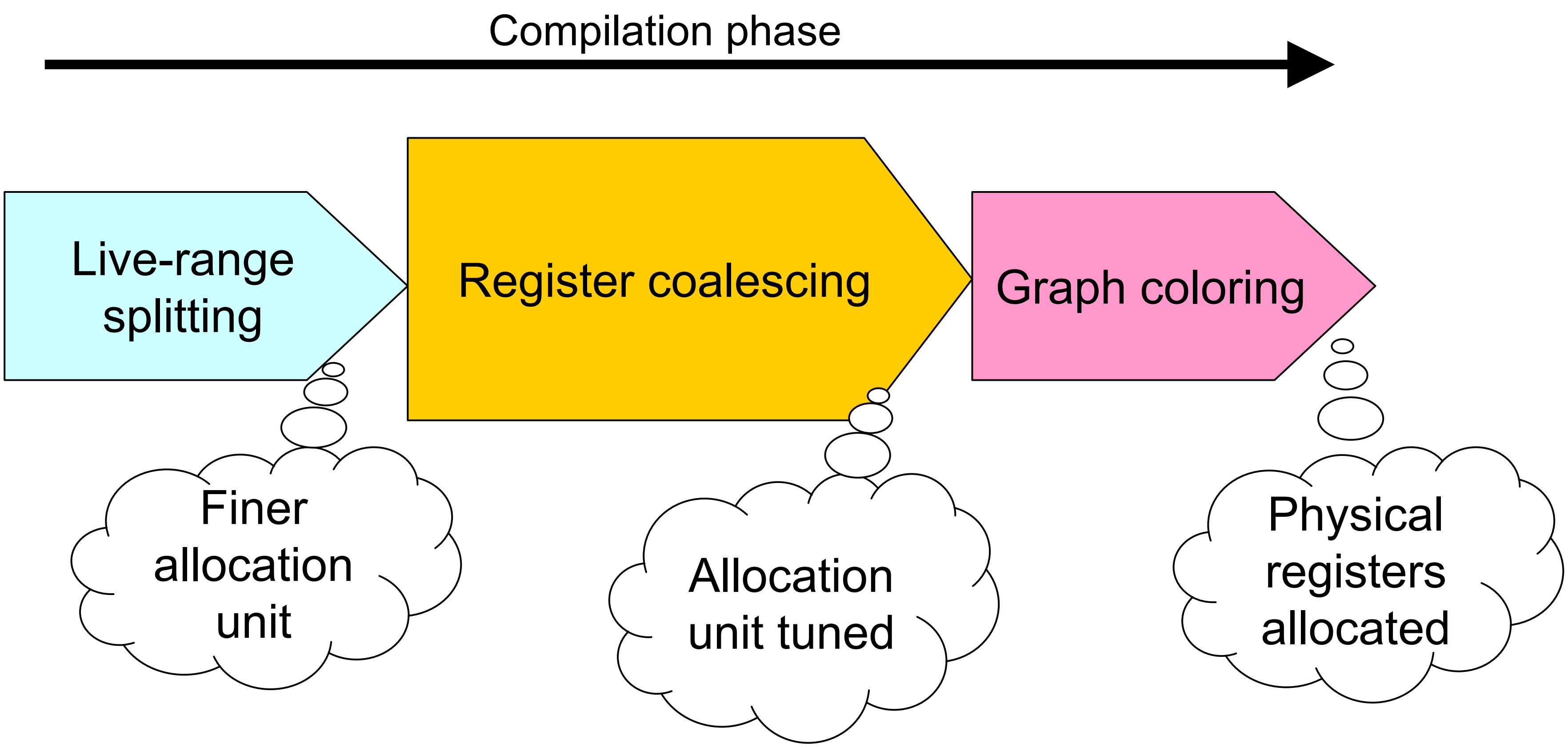
Rei Odaira, Takuya Nakaike, Tatsushi Inagaki, Hideaki Komatsu, Toshio Nakatani

IBM Research - Tokyo

CGO 2010 | April 27, 2010

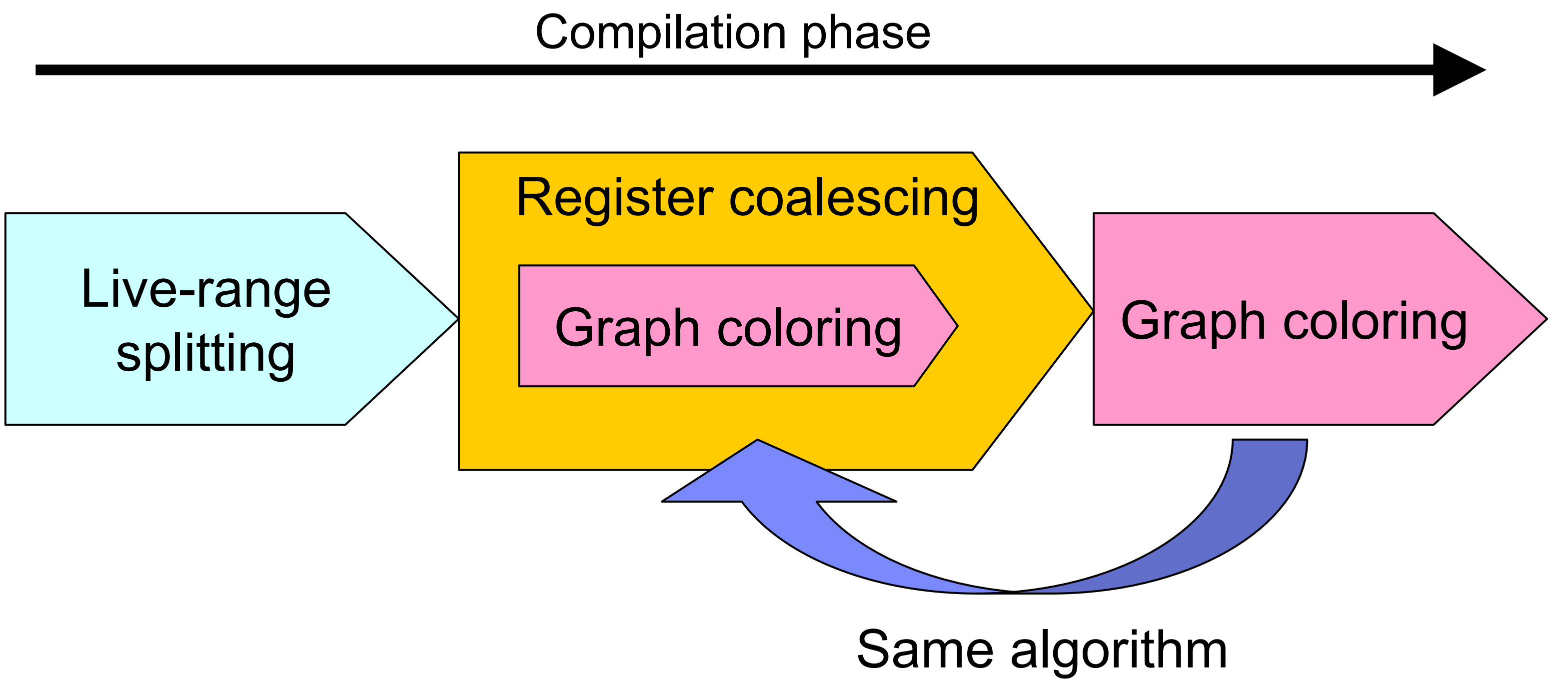
Register Allocation

- Goal: Reduce register spills!

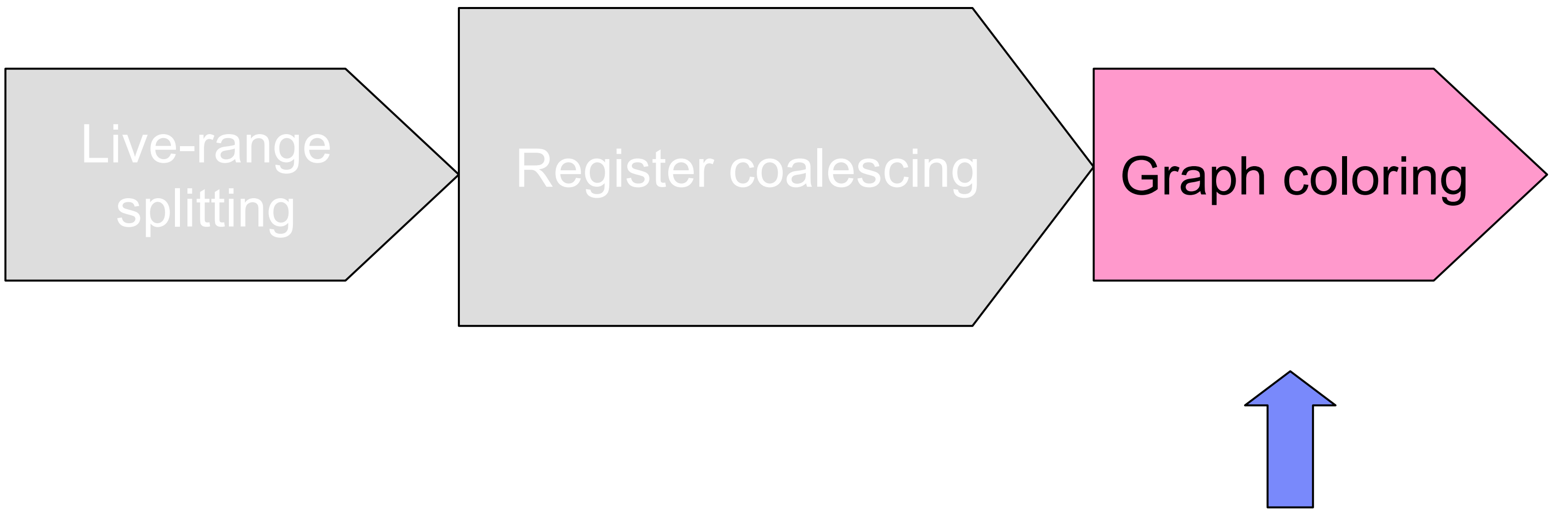


Our Approach: Use Graph Coloring in Register Coalescing

- Goal: Reduce more register spills!



Outline



Running Example

- Assign 3 variables to 2 physical registers.
  - A, B, and C
  - R1 and R2
- Need to spill one of the variables.

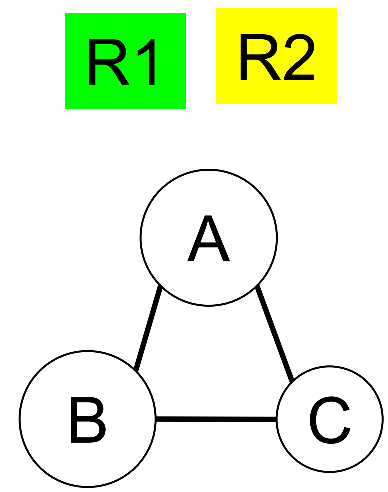
```
A = ...
B = ...
while (true) {
   C = ...
   ...= A + ...
   ...= C + ...
   if (...) {
      A = ...
      B = C + ...
  } else {
      if (B) {
          A = ...
          ...= B + ...
      } else {
         if (A > 0) break
      }
  }
  A = A + ...
  B = B + ...
}
```

Register Allocation as Graph Vertex Coloring

- Simple and powerful abstraction
  - [Chaitin et al., '81]

- Color = physical register
- Interference graph
  - Node = live range of a variable
  - Edge = interference between live ranges

```
A = ...
B = ...
while (true) {
    C = ...
    ...= A + ...
    ...= C + ...
    if (...) {
       A = ...
       B = C + ...
   } else {
       if (B) {
           A = ...
           ...= B + ...
       } else {
          if (A > 0) break
       }
   }
   A = A + ...
   B = B + ...
}
```



Which node to spill?

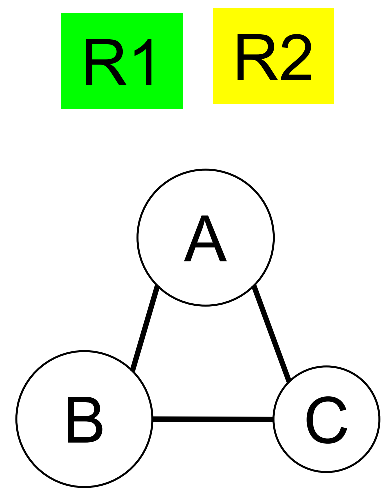
Calculating Spill Costs and Interference Degrees

```
A = ...
B = ...
while (true) {
    C = ...
    ...= A + ...
    ...= C + ...
    if (...) {
       A = ...
       B = C + ...
   } else {
       if (B) {
           A = ...
           ...= B + ...
       } else {
          if (A > 0) break
       }
   }
   A = A + ...
   B = B + ...
}
```

- Assume optimistic heuristics [Briggs, '94].

- Cost = frequency of accesses to a variable.
- Degree = how much a node restricts the coloring of its neighbors.

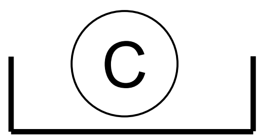
	Cost	Degree	Cost / Degree
A	30	2	15
B	30	2	15
C	20	2	10



Benefit of register allocation.

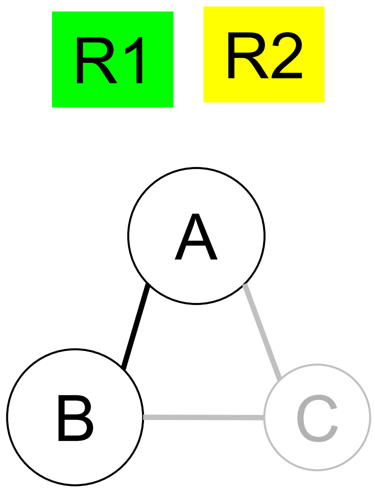
Simplifying Interference Graph

- Push the least beneficial node to a coloring stack.



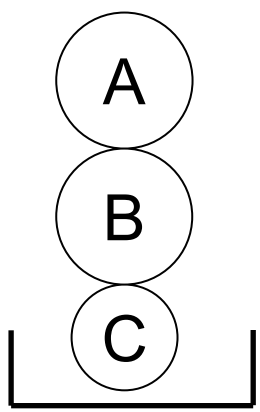
Coloring stack

	Cost	Degree	Cost / Degree
A	30	1	30
B	30	1	30
C	20	2	10

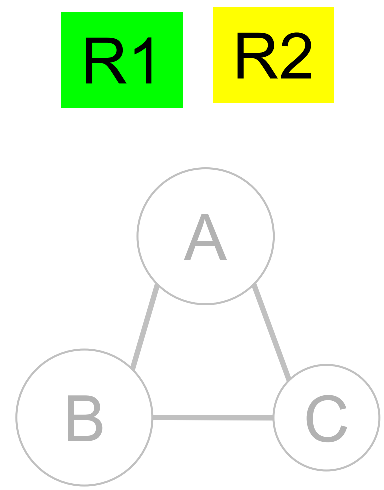


Simplifying Interference Graph

- Finished simplifying the graph.

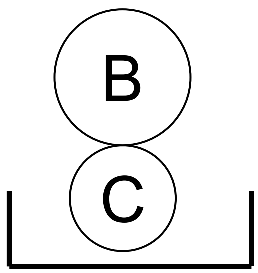


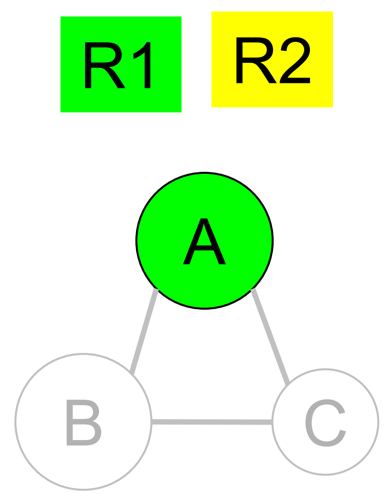
	Cost	Degree	Cost / Degree
A	30	1	30
B	30	1	30
C	20	2	10



Selecting Colors

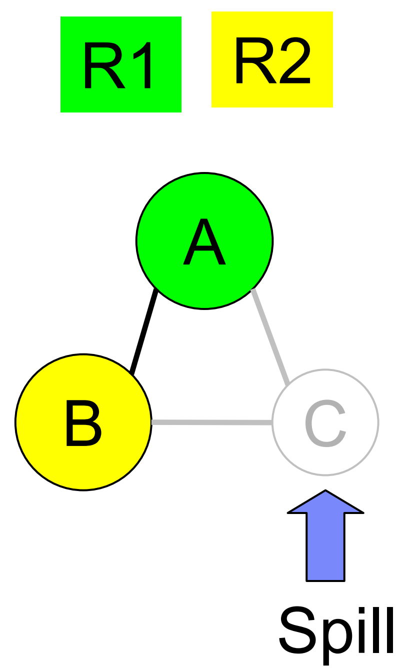
- Pop a node.
- Select a color that is not assigned to its neighbors.





Selecting Colors

- If no color is available, the node is marked for spilling.



Problem: Spill Everywhere is Costly.

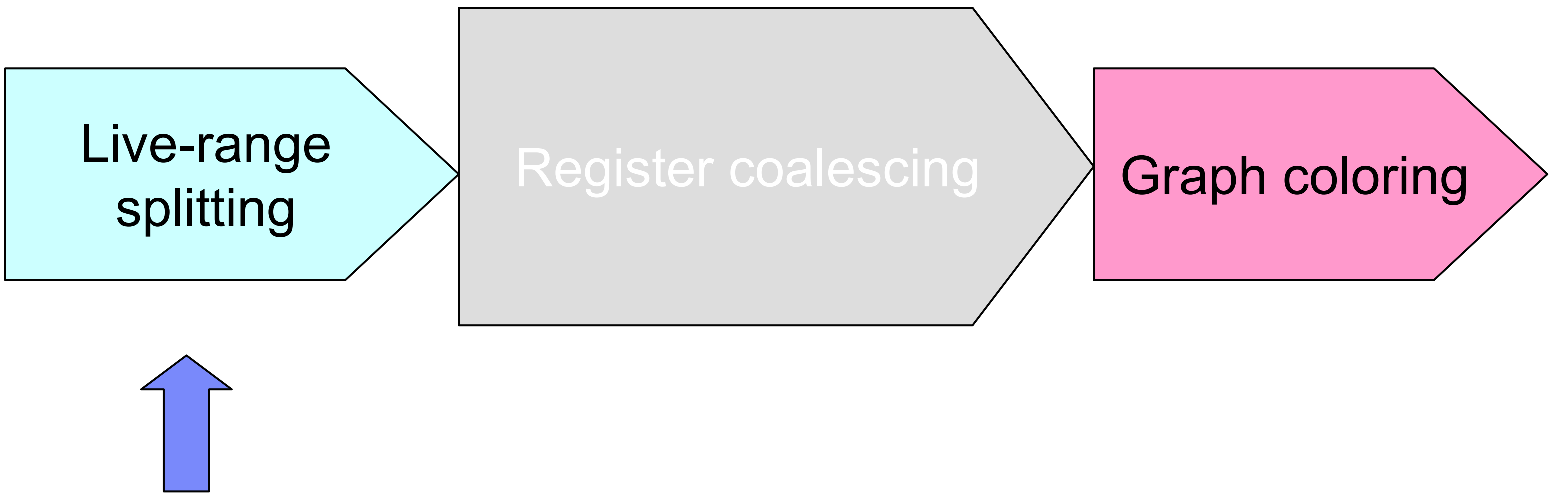
- Live range can be either:
  - Assigned to a single register, or
  - Entirely spilled to the stack.

- Spill can be further reduced:
  - By assigning only a part of a live range to a register, or
  - By assigning different parts to different registers.

→ Live-range splitting

```
R1 = ...
R2 = ...
while (true) {
   C = ...
   Store C to stack
   ...= R1 + ...
   Load C from stack
   ...= C + ...
   if (...) {
      R1 = ...
      Load C from stack
      R2 = C + ...
   } else {
      if (R2) {
         R1 = ...
         ...= R2 + ...
      } else {
         if (R1 > 0) break
      }
   }
   R1 = R1 + ...
   R2 = R2 + ...
}
```

Outline



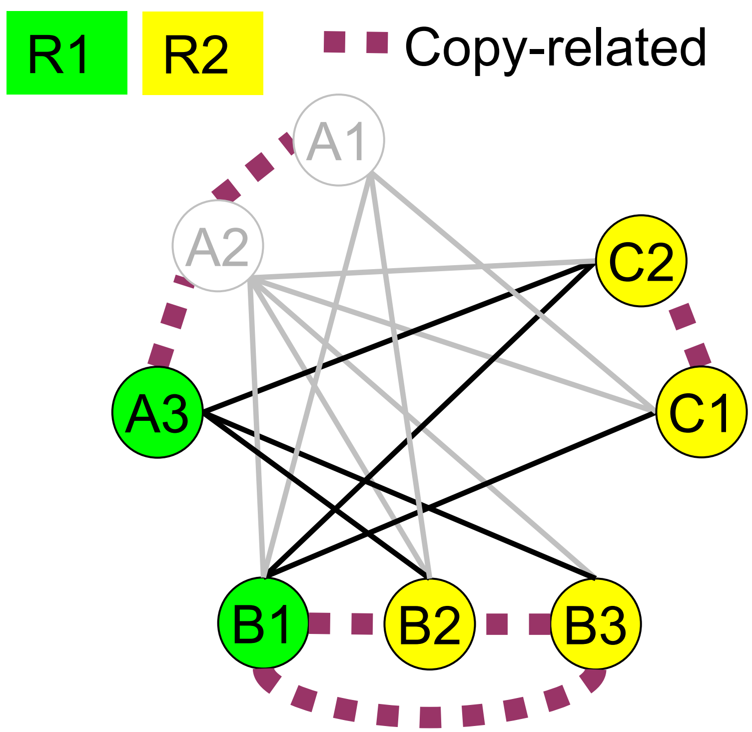
Live-range Splitting

- [Briggs, '92], [Kolte et al., '94], [Nakaike et al., '06], etc.
- Split live ranges into shorter *sub-ranges*: A1, A2, A3, etc.
  - Split sub-ranges are *copy-related*.
- Graph coloring can assign different colors to different sub-ranges.

```
A1 = ...
B1 = ...
while (true) {
    C1 = ...
    ...= A1 + ...
    A2 = A1
    ...= C1 + ...
    C2 = C1
    if (...) {
       A3 = ...
       B2 = C2 + ...
   } else {
       B2 = B3 = B1
       if (B1) {
          A3 = ...
          ...= B3 + ...
          B2 = B3
       } else {
         A3 = A2
         if (A2 > 0) break
       }
   }
   A1 = A3 + ...
   B1 = B2 + ...
}
```

In Reality, It's Not That Easy.

- Too large degrees and too small costs confuse the coloring heuristics.

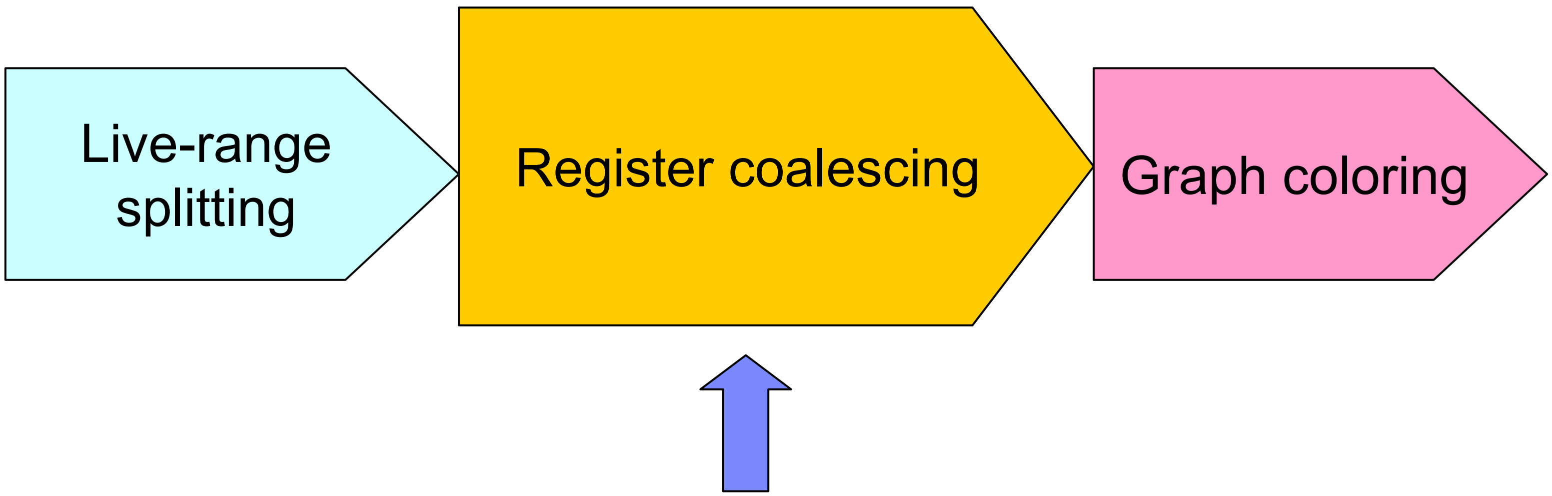


→ We need coalescing!

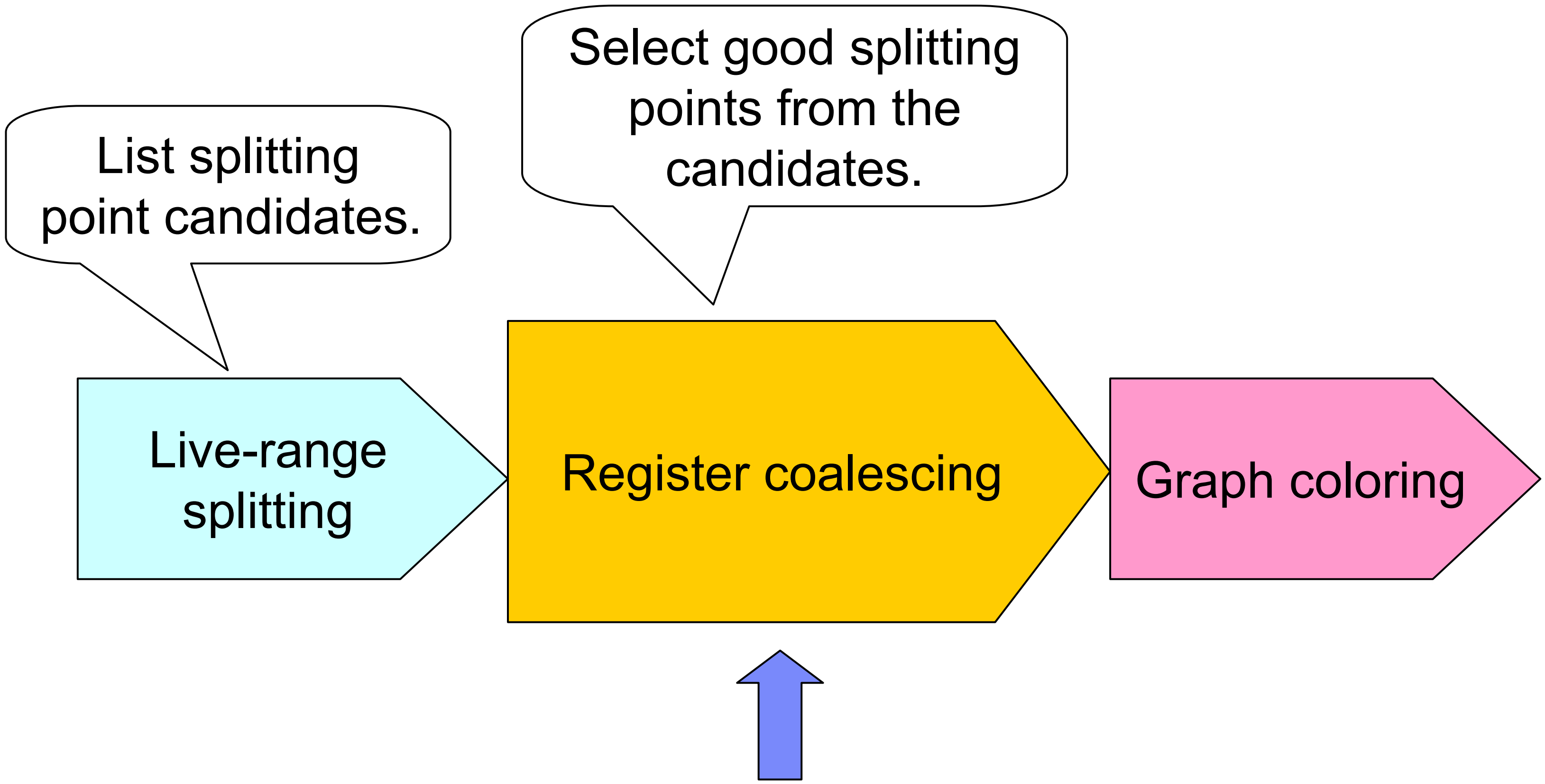
```
A1 = ...
Store A1 to stack
R1 = ...
while (true) {
   R2 = ...
   Load A1 from stack
   ...= A1 + ...
     ...= R2 + ...
     if (...) {
       R1 = ...
       R2 = R2 + ...
   } else {
       R2 = R1
       if (R1) {
          R1 = ...
          ...= R2 + ...
       } else {
         Load A2 from stack
         R1 = A2
         if (R1 > 0) break
       }
   }
   A1 = R1 + ...
   Store A1 to stack
   R1 = R2 + ...
}
```

No spill reduction by splitting.

Outline



Outline

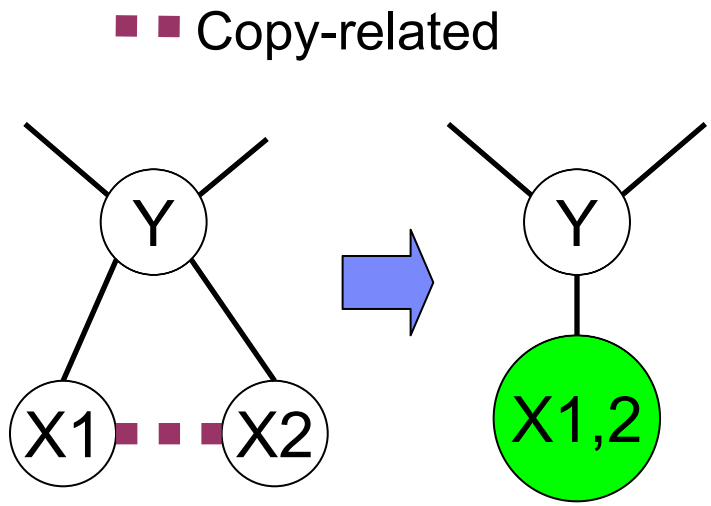

List splitting point candidates.
Select good splitting points from the candidates.
Live-range splitting
Register coalescing
Graph coloring

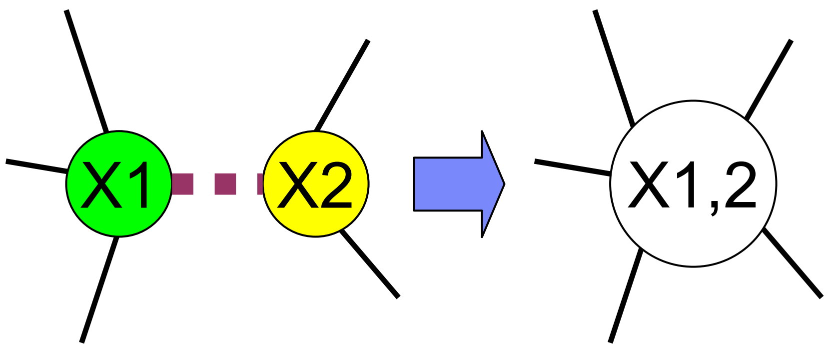
Register Coalescing

- Merge copy-related sub-ranges into a longer sub-range.
  – [Chaitin, ’82], [Briggs, ’94], [George et al., ’96], [Park et al, ’98]
  – Originally proposed to reduce copies.

To reduce spills, it has pros and cons.

- Pros: Coalesced node can become colorable.
  – Due to increased cost.
- Cons: Coalesced node can become uncolorable.
  – Due to increased degree.
- Depend on the number of common neighbors.

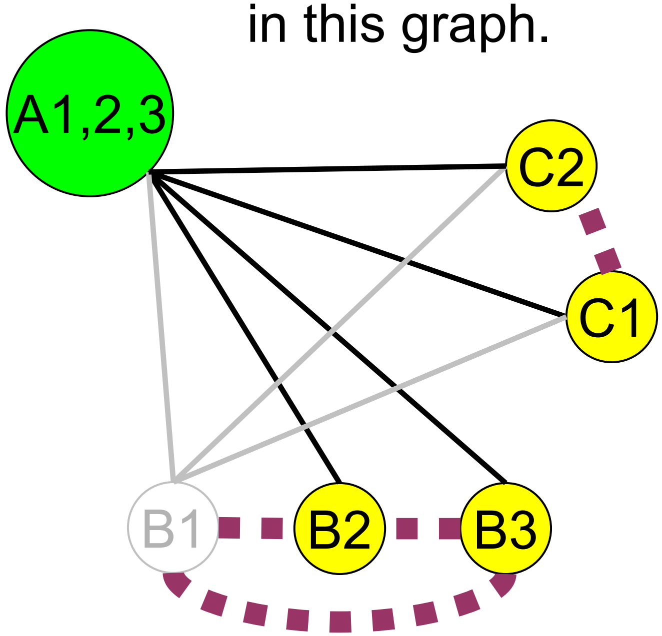




You Should Coalesce Those Nodes That Have Many Common Neighbors.

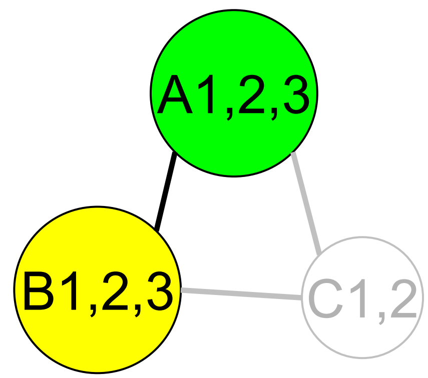
- As long as the coalesced nodes do not become uncolorable.
- No good criteria are known.

Coalesce those nodes that have 3 common neighbors.
→ Minimum spills in this graph.



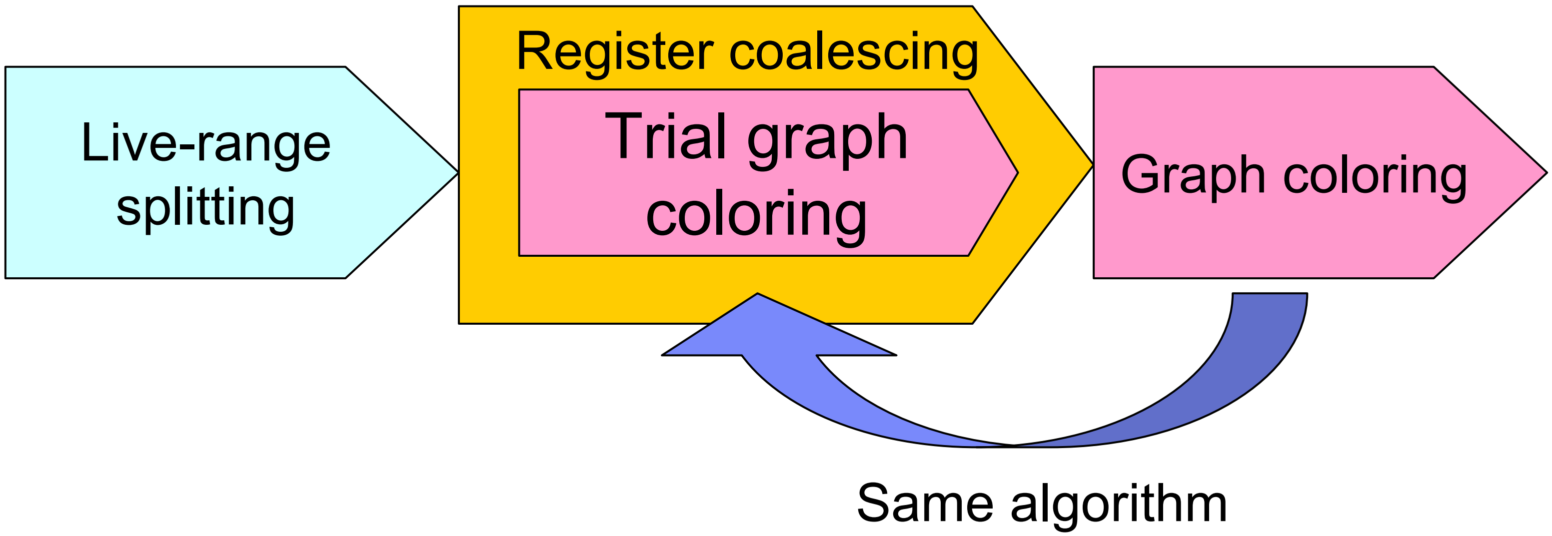
1 store,
1 load

Coalesce those nodes that have 2 common neighbors.
→ Revert to the original graph.



1 store,
2 loads

Our Approach



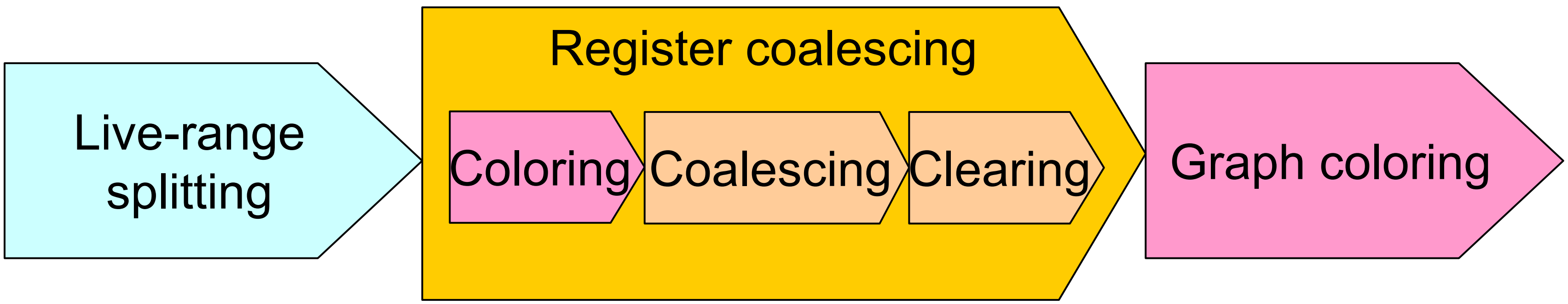
Our Rationale

Coloring results reflect the structure of a graph.

- Common neighbors
  ⇔ Likely the same color by trial coloring.
  – Common neighbors impose the same coloring restrictions.
- Can become uncolorable by coalescing
  ⇔ Likely different colors by trial coloring.
  – Interference prevents them from being assigned the same color.

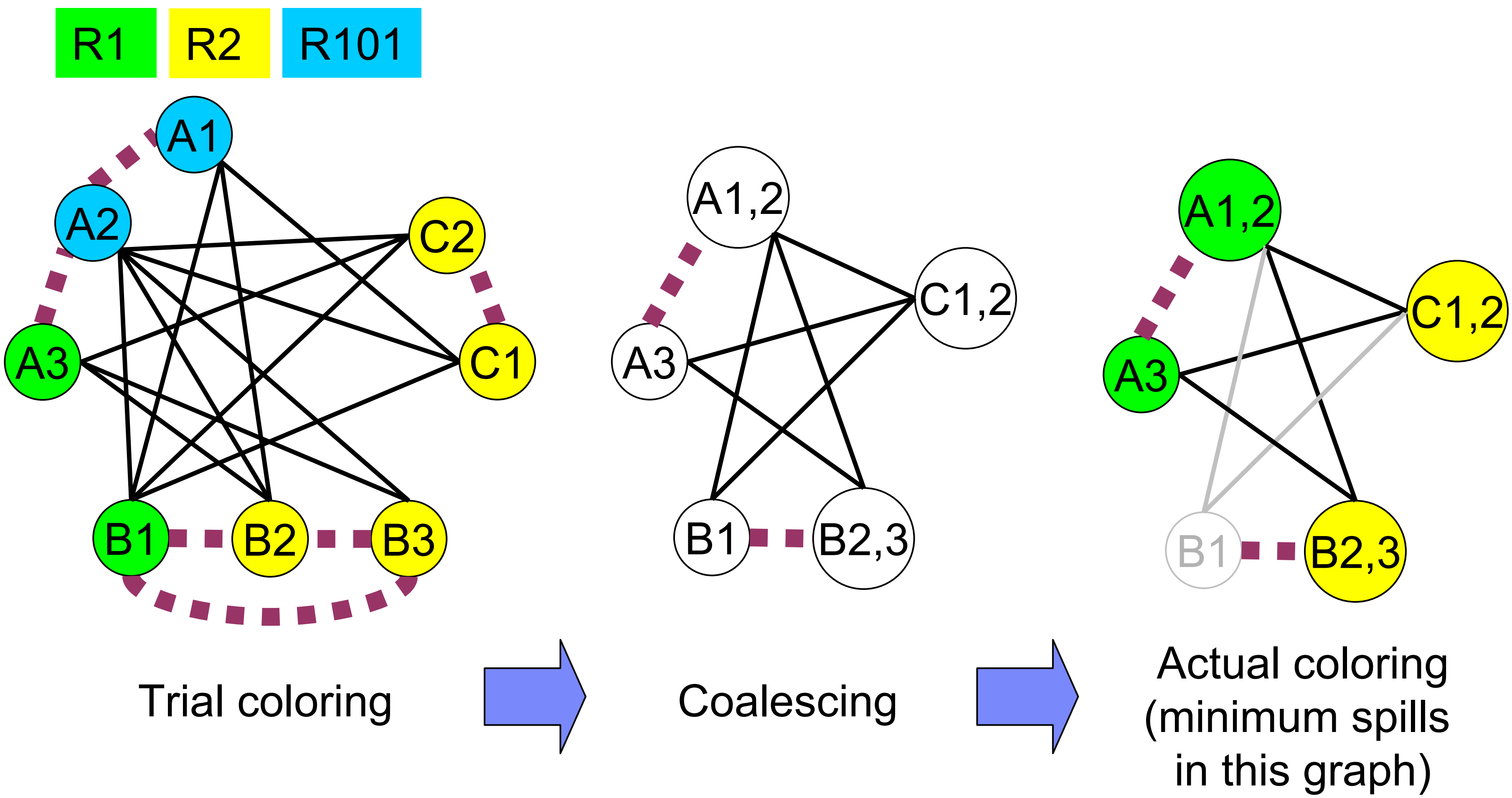
Coloring-based Coalescing

1. Do trial coloring.
2. Coalesce copy-related nodes that are assigned the same color.
3. Clear the colors.
4. Do actual coloring for register allocation.



Trial Coloring, Coalescing, and Actual Coloring

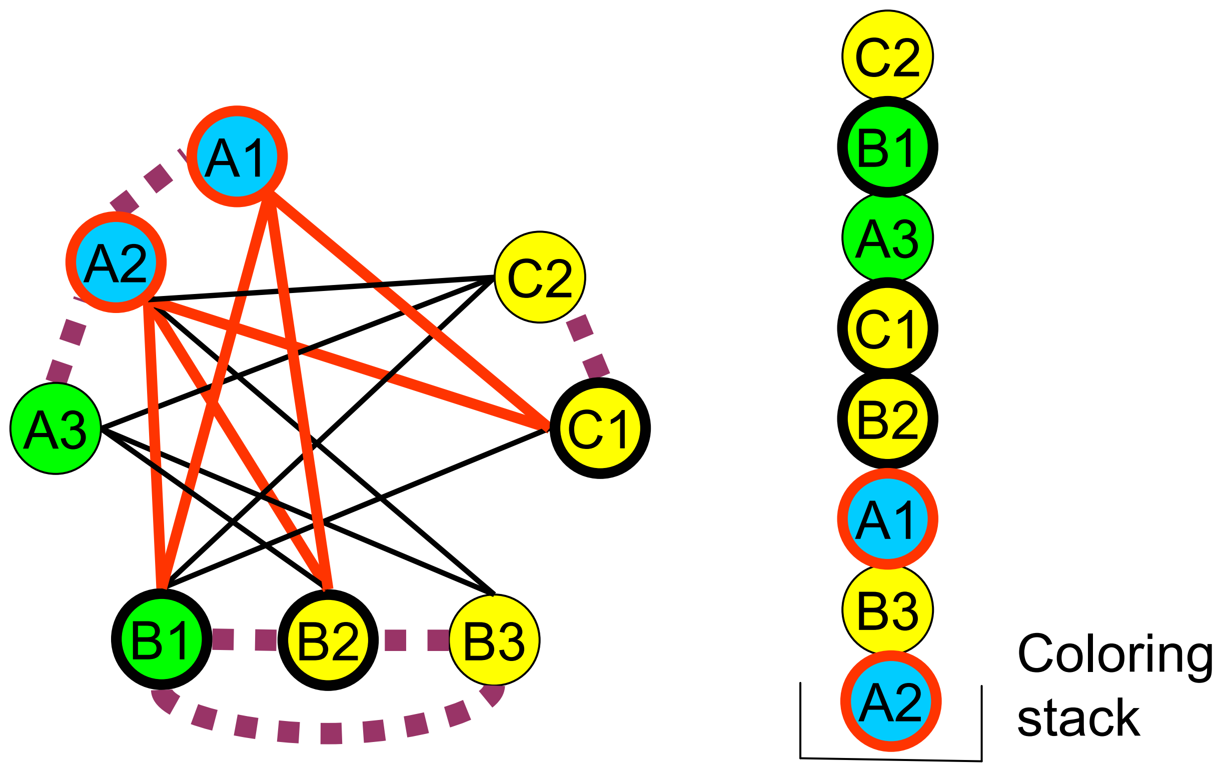
- Increase the number of colors on demand to color all nodes.



Two Key Points to Obtain Good Coalescing

1. Coalesce A1 and A2.
   – Because neighbors of A1 totally included in those of A2.

Trial coloring successfully assigns A1 and A2 the same color.
   – B1, B2, and C1 impose the same coloring on A1 and A2.

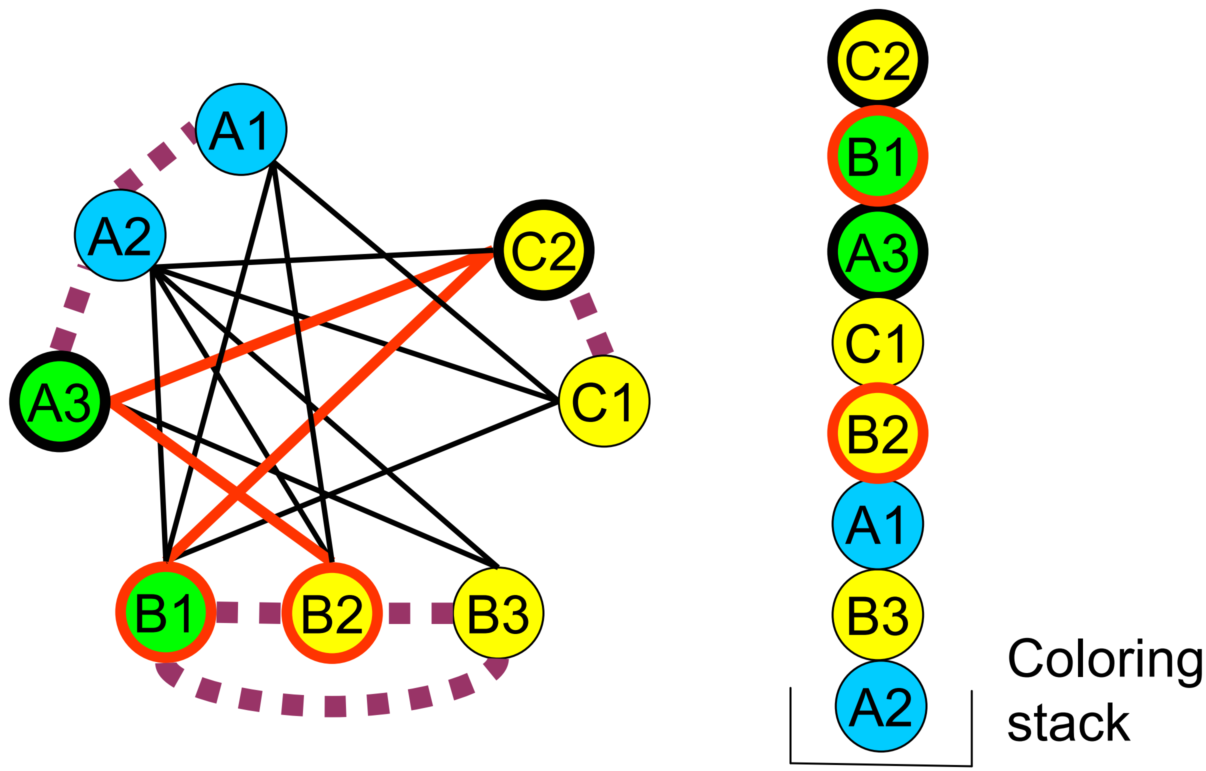


Two Key Points to Obtain Good Coalescing

2. Do not coalesce B1 with B2.
   - Because it could create a triangle, which is not 2-colorable.

Trial coloring successfully assigns them different colors.

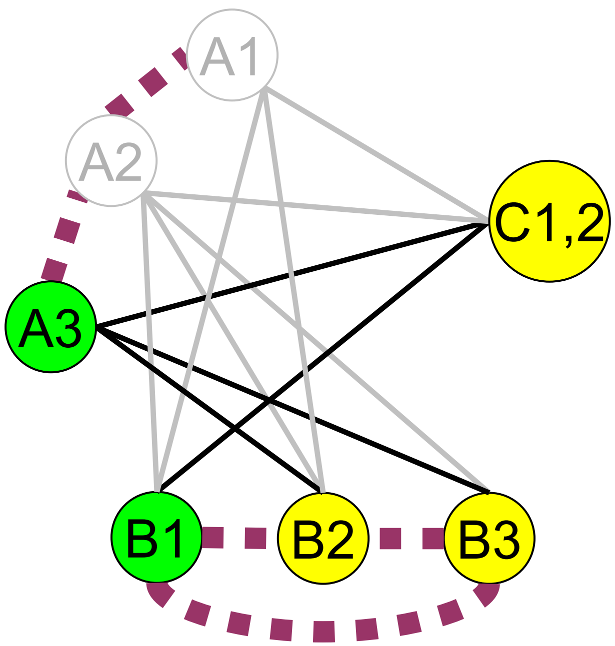
- Due to the 2-coloring of the chain of B1-C2-A3-B2.



Existing Algorithms are Too Conservative or Too Aggressive.

Iterated coalescing [George et al., '96]:

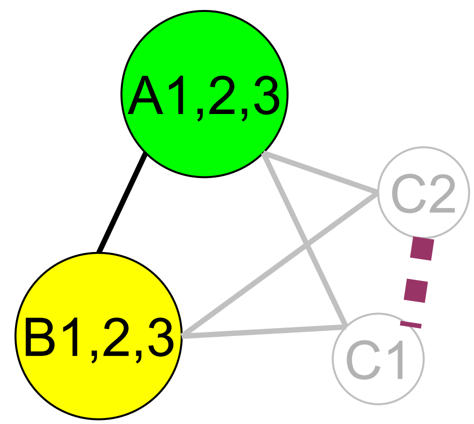
Must keep the colorability of coalesced nodes.



Optimistic coalescing [Park et al., '98]:

After aggressive coalescing, split again if uncolorable.

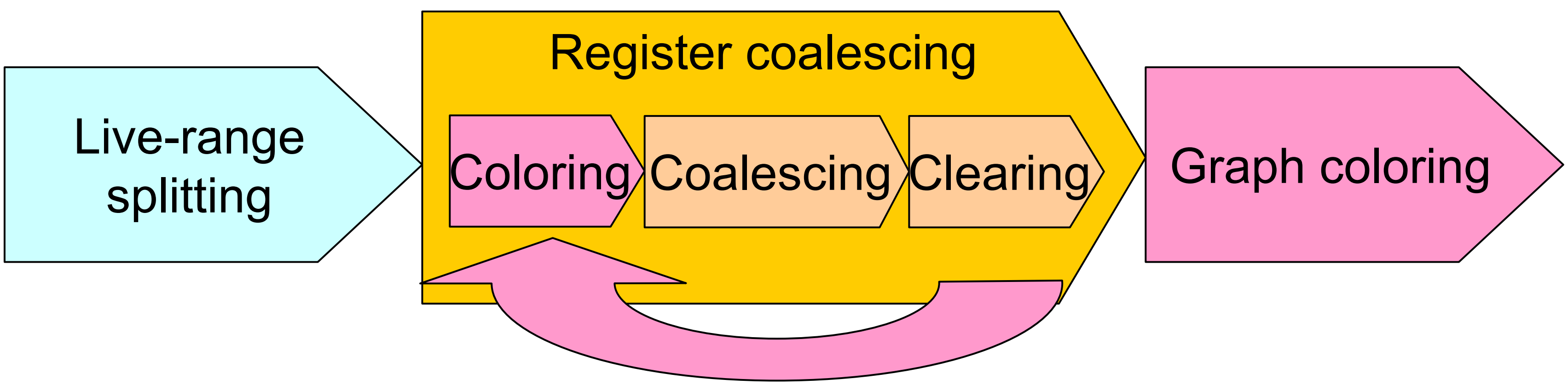
– But a colored node cannot be split again.



More Iterations Can Produce Better Results.

- But too many iterations can be harmful.
  - Increased spills.
  - Increased compilation time.

→ Need experiments.

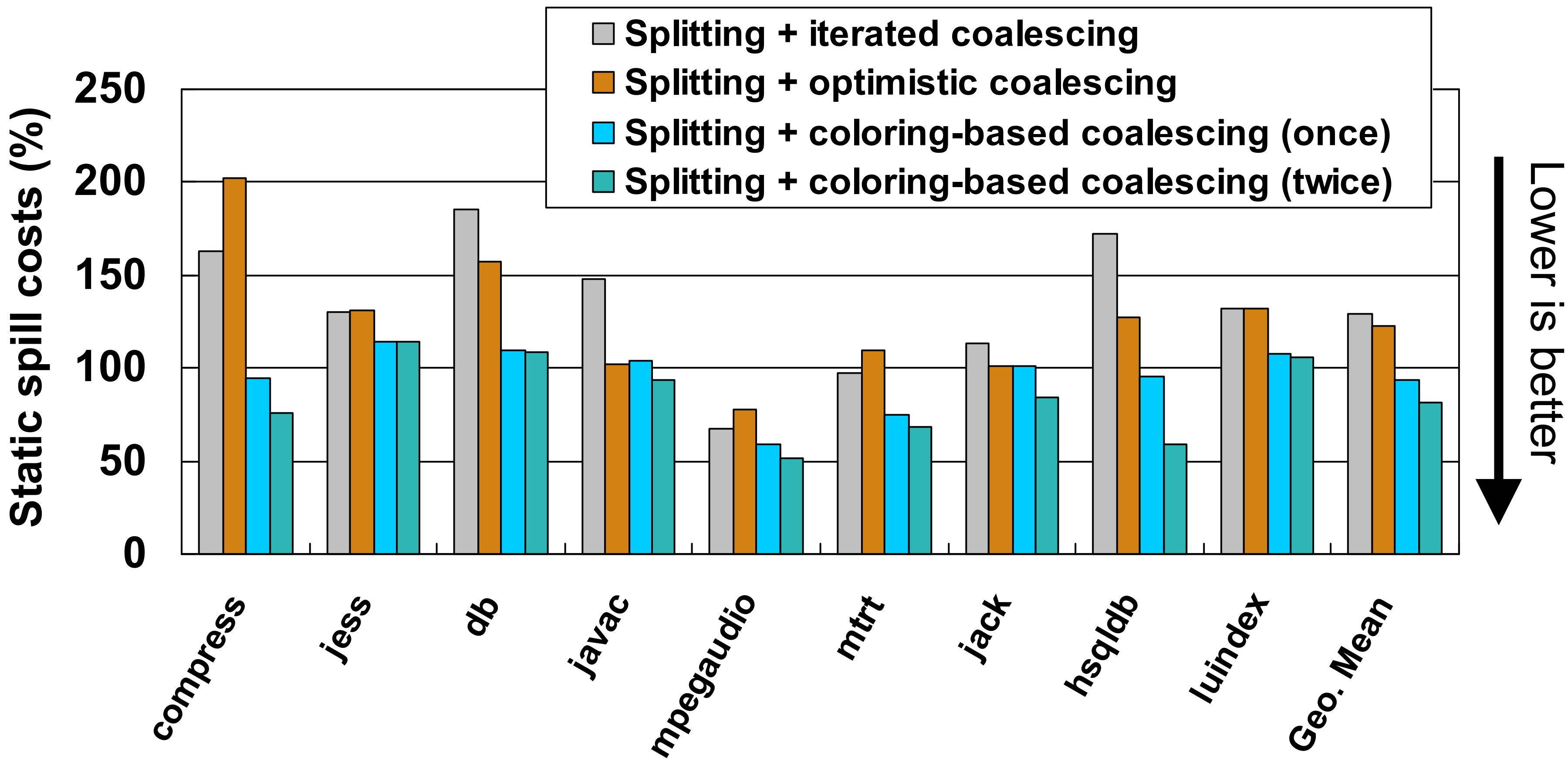


Experiments

- Environment
  - IBM J9/TR 2.4 JIT compiler
    - Implemented a graph coloring register allocator and the coalescing algorithms.
    - Implemented SSA-and-reverse-SSA-based live-range splitting [Briggs, '92].
  - IBM System z9 2094 / 4x 64-bit CPUs / 8GB memory / Linux 2.6.16
    - 16 integer and 16 floating-point registers.
- Benchmarks
  - SPECjvm98 and 2 larger benchmarks from DaCapo
- Spill cost calculation
  - Static number of uses and definitions, weighted by 10 in a loop
- Baseline
  - Graph coloring register allocator with iterated coalescing (no splitting)
- Compared approaches
  - Splitting + iterated coalescing
  - Splitting + optimistic coalescing
  - Splitting + coloring-based coalescing (once)
  - Splitting + coloring-based coalescing (twice)

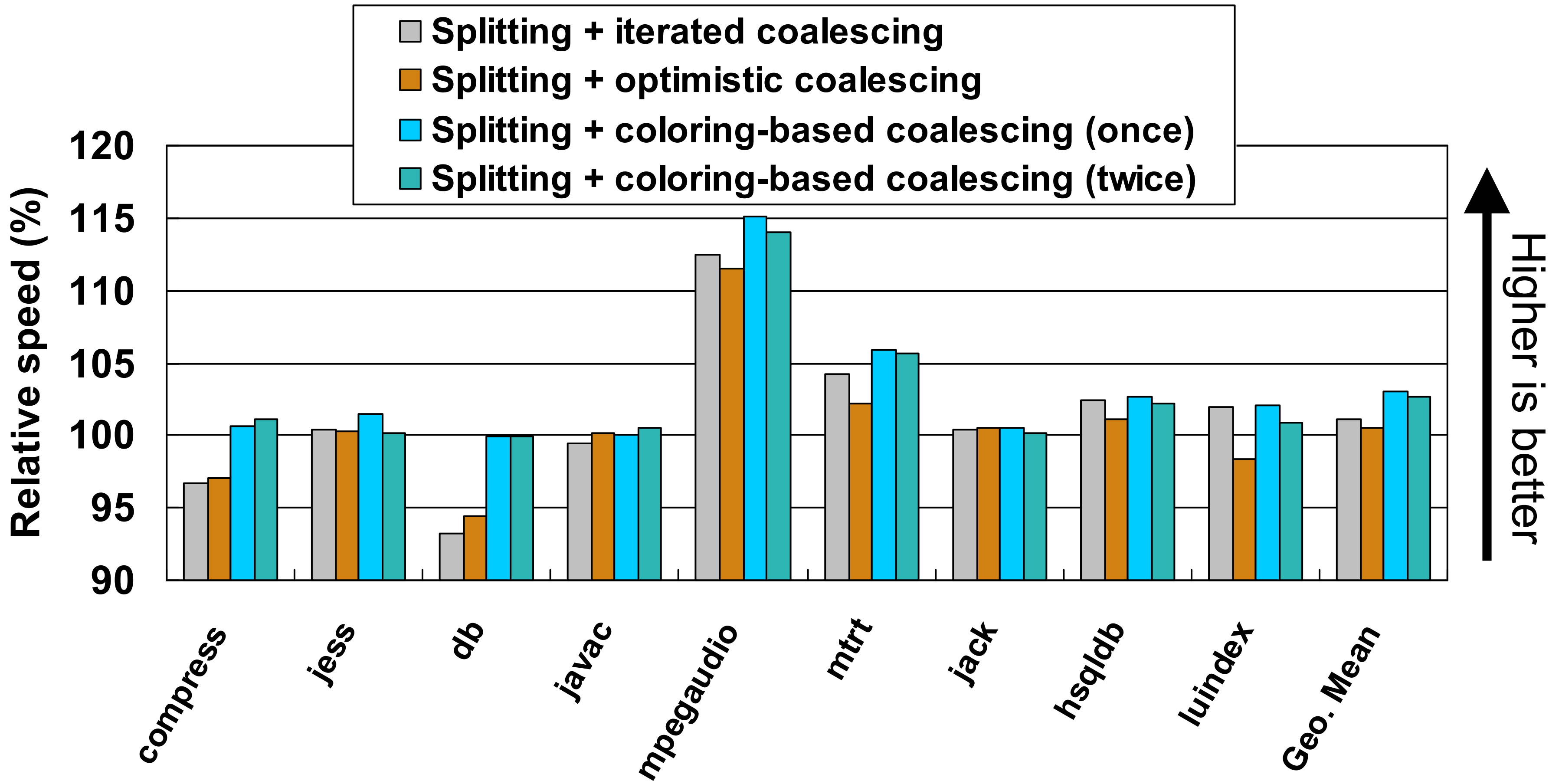
Static Spill Costs (100% = w/o Splitting)

- 6% reduction on average by coloring-based coalescing once.
  - 18% reduction by twice.
- More than 20% increase on average by the existing algorithms.



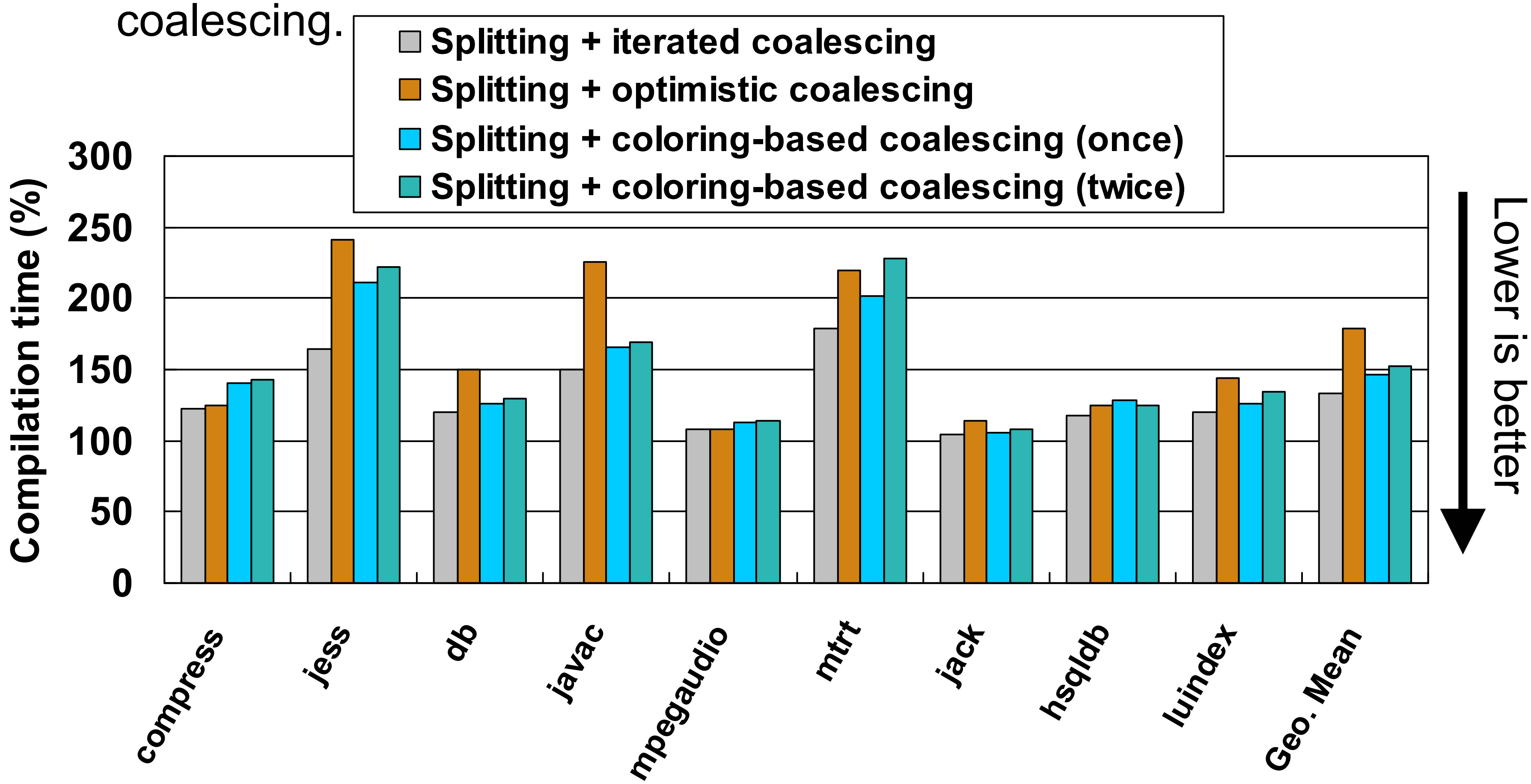
Execution Time (100% = w/o Splitting)

- JIT compilation time not included.
- Up to 15% and on average 3% speed-up.
- Up to 12% and on average 1% speed-up by the existing algorithms.



Compilation Time (100% = w/o Splitting)

- Increase mostly due to live-range splitting.
  - ~50% increase on average by coloring-based coalescing.
  - 32% increase by iterated coalescing, while 78% by optimistic coalescing.



Conclusions

Coloring-based coalescing effectively reduces spills.

- Simple
  - Just iterate an existing coloring algorithm.
- Powerful
  - Inspect the structure of an interference graph by trial coloring.

- 6% reduction on average in static spill costs.
  - 20% increase on average by the existing algorithms.
- Up to 15% and on average 3% speed-up
  - Up to 12% and on average 1% speed-up by the existing algorithms.

Thank you!

- Questions?

Backup

Static Copy Costs (100% = w/o Splitting)

- 13% reduction compared with iterated coalescing.
- 15% increase compared with optimistic coalescing.